A STUDY INTO THE PROBLEM OF DEADLOCK

Leslie Ray Conklin

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A STUDY INTO THE PROBLEM OF DEADLOCK

by

Leslie Ray Conklin

Thesis Advisor: G. L. Barksdale, Jr.

Approved for public release; distribution unlimited.

A Study into the Problem of Deadlock

by

Leslie Ray Conklin
Captain, United States Marine Corps
B. S., University of Missouri, 1965

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June, 1973

ABSTRACT

This paper presents a study of the problem of system deadlock with emphasis on the allocation of serially reusable resources. The initial area of concentration is on a definition of system deadlock and a presentation of proven methods of handling deadlock. After the discussion of current theory, two major operating systems are examined. An overall description of the functioning of OS/360 is presented followed by a detailed discussion of the procedures used by OS/360 to allocate serially reusable resources. The Michigan Terminal System is then discussed, outlining the techniques of resource allocation employed. The paper is concluded by correlating the methods used in OS/360 with those of MTS.

TABLE OF CONTENTS

I. INTRODUCTION 4
II. CURRENT THEORY 5
A. DEADLOCK DEFINITION 5
B. DEADLOCK STRATEGIES 7
1. Prevention 7
2. Detection 9
3. Crash 10
III. OPERATING SYSTEM/360 11
A. GENERAL ORGANIZATION 11
B. RESOURCE ALLOCATION 13
1. Job Selection 13
2. Region Management 14
3. I/O Device Allocation 14
4. Task Termination 15
C. SUMMARY 15
IV. MICHIGAN TERMINAL SYSTEM 17
A. GENERAL ORGANIZATION 17
B. RESOURCE ALLOCATION 18
1. Main Storage 19
2. Files 20
3. I/O Devices 23
C. SUMMARY 24
V. CONCLUSIONS 26
APPENDIX A 29
APPENDIX B 32
APPENDIX C 35
APPENDIX D 45
APPENDIX E 47
BIBLIOGRAPHY 49
INITIAL DISTRIBUTION LIST 51
FORM DD 1473 52

I. INTRODUCTION

In the present age of the computer there is an ever increasing demand for computer power. To some extent this demand has been fulfilled by using bigger and faster computers, but economic constraints have dictated that computers must also be efficient. The efficient utilization of computers is often achieved through multiprogramming.

In a multiprogramming environment several processes are active concurrently; consequently, there is now an interaction between processes as they compete for system resources. When several processes compete for resources it is possible that more than one process will require a given resource at some point in time. In this situation one process will be given control of the resource in question, and all other processes will be blocked until the resource again becomes available. If two or more processes are mutually blocking each other from further execution the situation is known as deadlock.

An investigation of the problem of system deadlock is undertaken in this thesis. A study of available literature on deadlock was conducted to determine what has been done in this area. An attempt is made to consolidate this information into a unified presentation of current theory. A second objective was to closely examine two major operating systems to determine which deadlock strategies have been employed in each and what methods were used in their implementation. The two operating systems selected for this study were IBM's OS/360 and the University of Michigan's Michigan Terminal System (MTS).

II. CURRENT THEORY

In studying the current literature on the subject of system deadlock it is evident that there are a limited number of authors in this area. It also appears that these authors are very much in agreement on the basic facts. The variance is encountered in nomenclature and representation methods. While one author [Habermann 1969] prefers a vector notation and uses matrix manipulation to define deadlock, another [Holt 1971] [Holt 1972] prefers a graphic representation and defines deadlock accordingly.

As a result of reading much of the current literature the author has developed a view of the deadlock problem which includes points presented by several writers. The following is a presentation of that view.

A. DEADLOCK DEFINITION

A simple instance of deadlock occurs when two processes, P1 and P2, use two common resources, R1 and R2. Assume that each process requests resources as they are needed and further assume that a resource cannot be released by a process which is waiting for a requested resource. Suppose process P1 has resource R1 allocated to it and requests R2; process P2 has resource R2 allocated to it and requests R1. This is a deadlock situation because by requesting the same two resources in reverse order each process has made a request which cannot be satisfied.

The simple case of system deadlock has been extended to include any number of processes and resources [Holt 1971] [Holt 1972]. Holt has presented a number of theorems which

define necessary and sufficient conditions for deadlock. For a complete description based on graph theory methods the reader should refer to [Holt 1971]. In general terms, deadlock will occur in any situation in which there is a closed loop of processes each requesting a resource(s) allocated to another process within the loop.

Although deadlock can occur from any type of process interaction, we shall emphasize the allocation of serially reusable resources since most system resources are serially reusable [Havender 1968]. Some examples are: storage media, system components, and information.

Storage media (such as main memory, tape, and disk or drum tracks) are considered to be serially reusable because they contain only one set of information at a time. When that set of information is no longer needed, the media can be used again to hold a new set of information.

System components (such as tape drives, disk drives, central processing units, and access mechanisms) are serially reusable because they are generally dedicated totally to one process at a time. One might argue that a disk drive may have a disk containing information for many processes and should therefore be considered shareable. This is true, but if the volume contains only information for one specific application it can be considered serially reusable.

Information is intrinsically shareable, unless it is defined as not shareable for security reasons. Information becomes serially reusable when it is being changed. When one process is updating a record, no other process can access that record until the update has been completed. This particular problem has been termed the "critical

section" problem and has been the subject of its own solutions [Dijkstra 1965].

Serially reusable resources can be characterized by the following five properties [Coffman 1971] [Holt 1971] [Holt 1972].

(1) There is a fixed number of units of each resource. When all of the units of a given resource have been allocated no further requests can be fulfilled.

(2) Each unit of a resource is either assigned to a particular process or is available for assignment.

(3) A unit of a resource can be allocated to at most one process at a time.

(4) A process can release any resource which it has acquired but not yet released.

(5) A resource assigned to a process can not be pre-empted by another process. Once a resource has been acquired it does not become available until it is released by the process which holds it.

B. DEADLOCK STRATEGIES

Any system in which processes compete for resources must have a strategy for handling the deadlock problem. These deadlock strategies belong to one of three classes: prevention, detection, or crash [Holt 1971] [Holt 1972].

1. Prevention

Under deadlock prevention the system is designed in such a way that deadlock is not possible. As would be expected, prevention algorithms tend to be overly restrictive. In many cases a process' request for resources

is not granted even though such action would not lead to deadlock.

The simplest prevention algorithm is to allow execution of but one process at a time. This method certainly prevents system deadlock, but it also prevents multiprogramming.

A second means of deadlock prevention requires collective requests for resources. A process must acquire all the resources it will use before it begins to execute. In this case a running process can never be blocked by an outstanding request for resources. As each process terminates it releases its resources and thereby insures that a blocked process will eventually acquire its resources. The obvious disadvantage of collective requests is that many resources may be allocated long before they are used, thereby denying their use by other processes.

A third method of deadlock prevention makes it possible to avoid allocating resources long before they are used. Under this method a process holding formerly obtained resources must release those resources before requesting an additional resource. If any of the original resources are still required they must be re-requested along with the additional resource [Havender 1968].

A more complicated method is known as ordered resources [Havender 1968] [Holt 1971] [Holt 1972]. Resources are partitioned into k classes, numbered 1,2,...,k. A process is allowed to request resources in class i only if it has not already been allocated resources in classes i, i+1, ..., k. Deadlock is prevented because two processes cannot request the same resources in different order as described in the initial example. Any process which holds a class k resource cannot be blocked and will

eventually release its resources. Consequently, any process holding class k-1 resources (and can only request class k resources) will receive its requested units and can procede to termination. This reasoning can be extended to verify that any process holding resources of any class will eventually receive all of its requested units and therefore will never reach a deadlock situation.

Using ordered resources, the more valuable a resource is, the higher its class should be. Resources in class k can be requested when actually needed and released as soon as they are not needed. Thus the higher the class of a resource the more efficiently it is used.

2. Detection

As the name implies, this strategy allows deadlocks to occur and takes corrective action when a deadlock is detected. Deadlock detection algorithms have been presented which examine the status of the system at any given instant to determine when corrective measures are required [Coffman 1971] [Holt 1971] [Holt 1972]. Once a deadlock has been detected the system can recover by terminating the deadlocked processes or by pre-empting resources from processes. Usually it is not necessary to terminate all of the deadlocked processes to recover. As soon as one of the processes acquires enough resources to begin execution the deadlock has been broken.

The obvious question which arises is which process is to be the one terminated or to have its resources pre-empted? It would be desirable to chose the process with the lowest associated cost. This cost could be defined as the cost of restarting the job and running it again to the point of termination or pre-emption. The difficulty encountered is determining this cost. It is not enough to

simply consider the amount of computer time used by the process up to this point, but one must also consider the integrity of the data involved. It is possible that terminating a process may result in a major effort to restore the data base to its original form.

Another method of determining restart cost would be to break down all processes into classes, each with an arbitrarily assigned cost. For example, in an academic environment the classes might be: student jobs, staff research jobs, and system jobs. In this case the students processing would be the first terminated or pre-empted.

This strategy can allow higher resource utilization than is possible when deadlock is entirely prevented. It is the preferred method in those systems where deadlock is infrequent and the cost of recovery is not prohibitive.

3. Crash

Under this strategy deadlocks are neither prevented nor detected by the operating system. It is the responsibility of the computer operator to determine when a deadlock has occurred and to take appropriate steps to remove it. The only possible advantage of this strategy is that it saves the time and space required by the prevention and detection routines.

III. OPERATING SYSTEM/360

The presentation on IBM's Operating System/360 (OS/360) is based primarily on information obtained from IBM program logic manuals. These manuals deal with the operating system functions at a very detailed level and hence were adequate for the purposes of this paper. The primary drawback of these manuals is that they are replete with abbreviations and consequently are difficult to comprehend.

The discussion on deadlock prevention in OS/360, as it pertains to serially reusable resources, is essentially a discussion of job initiation. It is during the initiation of a job step that all serially reusable resources are allocated, and thus where the deadlock prevention algorithms are encountered. Before a detailed description of the initiator is given, it is appropriate to briefly describe the general structure and functions of OS/360.

A. GENERAL ORGANIZATION

The operating system is divided into three major functional areas: job management, task management, and data management.

The primary functions of job management are: analysis of the job stream, overall scheduling, allocation of input/output devices, and direction of setup activities [Witt 1966]. The functions of job management are accomplished by the master scheduler and the job scheduler.

The master scheduler serves as a communication control link between the operator and the system. It processes

commands from the operator to the operating system and outputs messages from the system to the console typewriter.

The job scheduler consists of three primary elements: the reader/interpreter, the initiator/terminator, and the system writer. The reader/interpreter reads the input stream, converts the control statements into tabular form and places them into the job input queue. The initiator selects the highest priority job from the input queue, obtains a region of main storage, collects the required data sets, allocates I/O devices, and turns the job step over to task management. Upon completion of execution, the terminator directs the disposition of data sets and releases the I/O devices for allocation to other tasks. The system writer takes the output data sets produced by the job step from the output queue and puts them to the appropriate output device.

Task management controls the job step during its execution. The functions of task management consist of the fetching of required load modules; the dynamic allocation of CPU, storage space, channels, and control units on behalf of competing tasks; the services of the interval timer; and the synchronization of related tasks [Witt 1966].

The data management programs are primarily responsible for moving information between main storage and external storage and maintaining it in external storage. This is achieved by maintaining the system catalogue, providing various access methods and buffering techniques, processing volume labels, and protecting data via passwords [Clark 1966].

The above description of OS/360 was intended only to provide a basic view of the environment surrounding the system modules to be discussed later. For a more complete

description the reader should consult the above cited references and/or the IBM manual, Operating System Introduction [IBM 1971].

B. RESOURCE ALLOCATION

The discussion on resource allocation is presented on two different levels. The first level is general in nature and covers key points involving resource allocation and deadlock prevention. The second level appears as Appendix C and gives a detailed module by module description of the actions of the initiator. To aid those readers not familiar with IBM systems, Appendix A contains a glossary of IBM terms and Appendix B contains a description of relevant IBM system macro instructions.

The basic mechanism for controlling a resource is a pair of macro instructions, ENQ and DEQ. ENQ requests the use of a resource and specifies whether the control is to be exclusive or shared. DEQ is used to free resources obtained by the use of an ENQ.

The initiator is used to start both system tasks and job step tasks. To initiate a task the initiator retrieves job control information about the task and reserves system resources for the use of the task. After resource allocations are completed, the initiator passes control to the task. On task completion, the termination routine of the initiator releases all of the task's system resources. The actions of the initiator during the initiation of a typical job are discussed below.

1. Job Selection

The first function of the initiator is to select a job from the input queue containing jobs arranged by

priority within job class. To do this, the initiator uses the ENQ macro to obtain exclusive control of the queue control record. Once the initiator gains control of the queue control record, it examines it to determine if there are any jobs in the queue of the class serviced by this particular initiator. If so, the highest priority job within the class is selected. The initiator then issues a DEQ which permits access to the queue control record by other routines.

The initiator then determines how many steps are in the job and what data sets will be required. Next it prepares a parameter list for the ENQ macro specifying each data set name and whether its use is exclusive or shareable.

2. Region Management

The initiator then determines the amount of main storage required by the job step and issues the ENQ for its data sets. If all of the data sets cannot be obtained, the initiator issues a DEQ to free those which were obtained. This prevents possible deadlock between jobs with overlapping data set requirements. At this point the operator is given the option of retrying the ENQ, cancelling the job, or waiting for the data sets to become available.

When all data sets have been secured, the initiator then requests the region of main storage. When the region has been obtained, device allocation begins.

3. I/O Device Allocation

The operating system contains a table of unit control blocks. Each unit control block contains all of the information about a specific device that the initiator needs to make device allocations. In order to prevent multiple

;ses to the unit control blocks during allocation, the
.ator uses the ENQ to get exclusive control of them.
ENQ not only prevents simultaneous allocation of devices
ore than one job step, but also prevents the terminator
freeing any devices.

If allocation cannot be completed because of lack of
ces, the operator is given the option of cancelling the
making a device available which is offline, or waiting
devices to become available. If the wait option is
cted, the unit control blocks are DEQ'ed and the
iator waits until another task terminates, thereby
ing its devices. Allocation is then attempted again.

When all devices have been allocated the initiator
es a DEQ on the unit control blocks and the job step is
ied for execution.

4. Task Termination

When a task reaches normal completion or abnormally
execution, it is handled by the terminator, which acts
subroutine of the initiator. The terminator performs
required data set disposition and frees the I/O devices.
free the devices the terminator must use the ENQ to gain
lusive control over the unit control blocks. After all
ices have been freed, the terminator then issues a DEQ to
ease the unit control blocks. The initiator is now free
process the next job step.

SUMMARY

The above study of OS/360 has revealed quite clearly
t deadlock strategies have been incorporated into this
rating system. The designers of this system have chosen
prevent deadlock rather than to let it occur and then

take corrective action. It was probably their conclusion that, under the vast range of computing environments in which OS/360 is used, preventing deadlock was more practical than implementing detection algorithms.

The method of deadlock prevention used in OS/360 is a combination of two methods discussed at the beginning of this paper. The two methods are known as collective resources and ordered resources. The method of collective resources is applicable because a task must obtain all the resources it will use before it begins to execute. The method of ordered resources comes into play in that all resources are divided into three classes: data sets, main storage, and I/O devices. During allocation main storage is not requested until all data sets have been acquired; I/O devices are not requested until all data sets and main storage have been obtained.

IV. MICHIGAN TERMINAL SYSTEM

It was planned that this section of the thesis would contain a study of the Michigan Terminal System (MTS) comparable to that presented for OS/360. This study was to be carried out in connection with the implementation of MTS on the Naval Postgraduate School computer. However, MTS was undergoing revision at the University of Michigan and delays in its distribution precluded such a study.

In the absence of the current release of MTS and its associated documentation a much more limited effort was conducted. The primary sources of information were the computing center newsletter and memos published by the University of Michigan, various articles written by the designers of MTS, an assembly listing of a prior version of MTS, and telephone contact with the staff at the University of Michigan.

A. GENERAL ORGANIZATION

MTS is an operating system which offers both batch job and terminal usage. It was designed to take advantage of the special relocation hardware and multiple central processors offered by the IBM 360 model 67. In order to facilitate extension and modification, the system is organized as a set of independent components with well defined interfaces. The four primary components are the University of Michigan Multi-Programming Supervisor (UMMPS), the Paging Drum Processor (PDP), the Michigan Terminal System (MTS), and a highly modified version of the Houston Automatic Spooling Priority system (HASP).

UMMPS is a set of subroutines which run in privileged state with interrupts disabled. All entries to the supervisor are made through interrupts and are processed to completion before giving up the processor [Alexander 1971]. It processes all interrupts; does all physical input/output; allocates and schedules devices, main storage, and CPU's; and provides inter-task communication and protection [Boettner 1970] [Alexander 1971].

The PDP runs under the control of the supervisor and is responsible for maintaining external page storage and for moving pages between main storage and auxiliary storage [Boettner 1970] [Alexander 1971]. The auxiliary storage used is one or more drums or disks; these devices are under the complete control of the PDP.

MTS is a re-entrant program which is activated once for each terminal in the system and once for each running batch job [Boettner 1970]. In effect each user in the system has his own private copy of MTS. It provides device support routines, performs file handling functions, does command processing, and interfaces input/output with HASP.

HASP is a non re-entrant program which controls the card readers, punches, and printers on the system and maintains the input/output spool queue. The modifications to HASP have transformed it into a batch monitor. As such it generates MTS batch tasks and starts them executing. The number of batch jobs executing at any one time depends upon the system load [Boettner 1970].

B. RESOURCE ALLOCATION

Our discussion of allocation under MTS will be based on the type of resource being allocated, specifically, main storage, files, and I/O devices.

1. Main Storage

The task of moving pages between main storage and auxiliary storage is shared by UMMPS and the PDP. UMMPS determines which pages are to be read into memory or written out to drum and the PDP does the reading or writing. Communication between UMMPS and the PDP is accomplished by placing a Page Control Block (PCB) on one of four queues: the Page In Queue (PIQ), the Page In Complete Queue (PICQ), the Page Out Queue (POQ), or the Release Page Queue (RPQ) [Alexander 1971]. An explanation of these queues and the PCB is contained in Appendix D.

The PDP utilizes five supervisor calls in its normal operation. These supervisor calls, described in Appendix E, are: Get Real Page (GETRP), Free Real Page (FREERP), Get Write Page (GETWP), PDP Wait (PDPWAIT), and Get Queues (GETQS) [Alexander 1971].

When the supervisor determines that a page must be read into main storage it will place the PCB for that page on the PIQ. Likewise, the supervisor will place pages which have been released by their tasks on the RPQ and pages which could be removed if necessary on the POQ. After placing a page on the PIQ the supervisor will start the PDP, if it is idle.

The PDP will call the GETQS subroutine which will return all PCB's on the PIQ and the RPQ. It next calls GETRP to obtain a main storage block to read each page into. In the event that no block is available, the GETWP routine will return pages to be written and main storage will become available.

When the PDP has obtained a block for each page to be read in, it constructs a channel program which will read the required pages into main storage. The channel program is generally constructed to accommodate nine pages at a time. If all nine slots are not filled by read requests, the PDP will call GETWP to attempt to get enough pages to fill the unused slots with writes.

When the channel program has been constructed for all the reads and writes, it will be queued for execution. During execution of the channel program, a program controlled interrupt is generated as each page transfer is completed. These interrupts are not queued and are lost if not serviced immediately. At the end of the channel program, however, an interrupt occurs which is queued. Interrupts of either type signal the PDP. The PDP then places the PCB's for pages read on the PICQ and calls FREERP to release the main storage blocks for all pages written. The supervisor will detect the pages on the PICQ the next time it is entered and will restart the tasks waiting for them.

The PDP also has the ability to detect unnecessary reads and writes and eliminate them. For example, if a page has not been changed since it was last read in, there is no need to write it back out again. By the same reasoning, if a page is requested again before it has been freed by the PDP, the copy already in main storage can be used.

2. Files

MTS supports the concept of shared files. After a user creates a file he has the ability to specify which other users may have access to this file and what type of access they may have. The owner of the file can limit the

access to himself, to other users by individual user identification numbers, to other users by project number, to all other users, or to any combination of the above [U of Mich 1972b]. As the owner of the file specifies who has access he also defines the extent of the access. The various degrees of access to a file are: the ability to read from, the ability to add to but not change existing information in the file, the ability to change or delete but not add to, the ability to truncate or renumber, the ability to destroy or rename, the ability to modify the right of access, or a combination of the above [U of Mich 1972b].

The system controls the concurrent use of shared files by maintaining a system wide shared file table. This table contains an entry for each file currently in use which describes how and by whom each file is being used. The system also requires that a file must be locked in the appropriate manner before any operation can be performed on that file [U of Mich 1972a].

MTS provides three level of locking. The lowest level is locking a file for reading. The second level is locking a file for modification, which includes adding, changing, deleting, truncating and renumbering. The highest level of locking a file is for destroying; this includes renaming and changing the rights of access. At each level of locking the user also has the file locked for the operations allowed by any lower level of locking [U of Mich 1972a]. As is normally true, any number of users may have a file locked for reading, but when a user has a file locked for modification or for destroying he must have exclusive use.

When a user requests a file, MTS determines whether or not he has been granted access by the owner of the file and, if so, has he been granted the degree of access

requested. If the answer to the two above questions is afirmative, MTS then consults the shared file table to determine if the required level of locking can be made without violating the rules requiring exclusive control. If so the file is locked as requested.

If another task already has the file locked and the current request cannot be honored, MTS will enqueue the task on the file and put it in a wait until the file can be locked. Before enqueueing the task, MTS will ensure that placing the task on the queue will not result in a deadlock situation. Perhaps this can best be illustrated through the use of an example.

Consider a situation similar to the one presented on page five. Assume that task T1 has file F1 locked for reading and that task T2 has file F2 locked for reading. Now suppose task T1 requires file F2 to be locked for modification. MTS will determine that F2 is locked for reading and will queue T1 to wait until T2 has completed its use of F2.

If at this point T2 now has a requirement to modify F1, MTS will determine that putting T2 on a queue to wait for file F1 would result in deadlock. Therefore, it will not place task T2 on the queue, but will instead indicate that an error condition exists.

What happens as a result of the error condition depends upon the type of job involved. If T2 is a batch job, it will be terminated, releasing all of its resources. It will then be returned to the HASP input queue in a hold status to be released by the operator to rerun at a later time. If T2 is a terminal job, an error message will be sent to the user indicating the deadlock situation and he must decide what corrective action is to be taken.

3. I/O Devices

MTS in general divides its I/O devices into two classes. One class consists of unit record devices such as the card readers, punches, and printers; the other class consists of all other devices.

The unit record devices are controlled by HASP. When a task requires a unit record device it is assigned a pseudo-device. As the task reads from or writes to this pseudo-device it is actually reading from or writing to the HASP spool queue. HASP in turn performs the I/O operation to a physical device as devices become available, assigning the tasks on a priority basis.

Other devices can be allocated either before the task begins execution or during the execution of the task. The time allocation takes place is determined by the method employed by the user in requesting the device. If the user prefers that allocation take place before execution begins, he must use a system command which invokes the MOUNT program. If allocation is to take place as needed during execution, the MOUNT program is called as a subroutine.

In either case the MOUNT program is supplied with the type of device required and the name of the file to mounted on the device. The MOUNT routine issues a message to the console operator requesting a device by type and indicating the volume to be placed on that device. If the operator can determine that there are not enough devices available to satisfy the allocation request, he can at this time so indicate. By doing so he forces a batch job to be returned to the HASP input queue, or forces a terminal user to decide on an alternate course of action.

If devices are available, the operator responds with a specific device address. The MOUNT routine then issues an SVC to enter the UMMPS allocation routine which attempts to allocate the named device. If the device is available, it is allocated and the operator is given confirmation.

In the event that the device named by the operator has already been allocated, is inoperable, or has been varied offline, an error condition is indicated. The response to this error condition is the same as that previously described for files.

C. SUMMARY

The above study of MTS does not reveal a clearly defined deadlock strategy. The allocation method encountered is different for each type of resource involved and there does not appear to be any interrelationship between them. Consequently, the deadlock strategy is at best difficult to pinpoint.

Of the three deadlock strategies described in the early part of this paper, prevention is the most appropriate to be associated with MTS. However, MTS does not employ any of the methods of prevention previously discussed. In spite of this fact, there is evidence supporting the choice of deadlock prevention as the strategy incorporated by MTS.

The most convincing evidence is displayed by the locking technique used in the control of files. Before a task is queued on a file, the system checks to see if such queueing would result in deadlock. If so, the task is not queued and deadlock is prevented.

A more subtle prevention technique is encountered in the allocation of main storage. Paging of main storage ensures

that no task will be blocked permanently from execution because it cannot get memory space. If a task has memory space and is blocked while waiting for another resource, its pages will gradually migrate to auxiliary storage enabling other tasks to get space.

The deadlock prevention method used in the allocation of devices is less sophisticated than that used for main storage or files. In fact it can be considered as no method at all. The issue is avoided by forcing tasks which cannot allocate devices to be rerun at a later time, in the hope that allocation will be possible when rerun is attempted.

V. CONCLUSIONS

Studying the resource allocation techniques of OS/360 and MTS has revealed strengths and weaknesses in both systems. Possibly at some time in the future an operating system will be developed which incorporates the good features and eliminates the poor features of both systems.

An advantage of OS/360 over MTS is that OS does not require interaction with the operator during the normal allocation of devices. In a production environment where a majority of the I/O is not of the unit record type, much time would be lost waiting for the operator to choose a device and type in his response.

Both systems use undesireable methods when the allocation of all requested resources is not possible at the time of the request. Under OS the operator is given an option which includes cancelling the job or waiting for resources. If he chooses to cancel the job, it is flushed out of the system. Why? Since OS requires collective allocation of resources, the task has not yet begun to execute. It would seem logical to place the job back on the input queue and attempt to run it later. If the operator is reluctant to cancel the job and puts it into a wait the initiator is tied up during this period and cannot begin another job step. This is especially dangerous when the operator has no idea how long this wait may be. If he is not careful he may seriously degrade the performance of the system. For example, this could happen if he put more than one job into a wait when the resource required was being held by a job which will run for several hours.

Under MTS the situation is different. When a task is put into a wait condition it does not hamper the execution of the remaining tasks and will not prevent other tasks from beginning to execute. The drawback comes when a task is terminated and returned to the input queue. Since MTS allows allocation of resources at any time, it is possible that a major file modification could have been in progress or completed at the time the job was requeued. Arbitrarily rerunning such a job can cause serious file integrity problems. It becomes the user's responsibility to avoid rerun under this type of condition.

In the areas of memory and file management, MTS utilizes the most effective techniques. Under OS a task requires a contiguous area of main storage to execute. The rapid reallocation of memory that occurs during multiprogramming tends to fragment the available memory space. Often a region cannot be allocated, even though enough free space exists, because this space is not contiguous. The paging method of MTS alleviates two problems: first, the entire task does not have to be in main storage at one time, and second, the portion of the task which is in main storage does not require a contiguous area.

Under OS the only files which can be used concurrently are direct access files. If a user wishes to share a disk file with others he has no way of restricting it to specific users and, even then, his restriction is only in effect during the time that the file is allocated to him. MTS not only gives the owner of a file the ability to share it with other users, but also the ability to restrict the type of access granted to each. Also, these restrictions are permanently associated with the file and are not in effect just while the owner has the file allocated to one of his tasks.

As far as system deadlock is concerned, it appears that both systems prevent deadlock but the methods used can at times be very costly. Unfortunately it is much simpler to point out the shortcomings of either system than to present an alternative solution.

APPENDIX A

This appendix contains explanations of IBM terms used in discussing resource allocation in OS/360.

Allocate Work Table: The Allocate Work Table is constructed by a submodule of the I/O Device Allocation routine. The table contains an entry for each DD statement in the input stream. Each entry contains information which describes the data set and information that is used in allocating a device to it. This information includes the number of volumes in the data set, the number of devices requested, the number of devices allocated, the number of devices shared, the DD number, the number of devices available for allocation, and a bit pattern representing which devices are eligible for allocation.

Job Control Table: The Job Control Table is created by the interpreter and placed in the input queue. It contains information from the JOB statement, job status information, and pointers to other tables in the job's input queue entry. It also contains the job region size, number of Step Input/Output Tables, and whether the job is to have checkpoint/restart [IBM 1972a].

Link Pack Area: The Link Pack Area is a region at the top (highest addresses) of main storage. This area contains useful re-entrant routines and tables which remain core resident. These routines and tables are generally highly used and having them

core resident eliminates the time required to read them in from disk or drum. Which routines are to be included in the Link Pack Area is determined before the system is generated [Flores 1973].

Queue Control Record: The Queue Control Record is made up of a series of pointers to entries in the queue. One entry points to the highest priority job in the queue. The remaining 15 entries point to the last job entered into the queue for each priority. (The elements within the queue are chained from highest to lowest priority, with jobs within a priority arranged in order of entry.) The organization of the Queue Control Record facilitates removing the highest priority job from the queue or entering jobs to the end of each priority string [IBM 1972a].

Step Control Table: The interpreter constructs a Step Control Table for each step of a job. This table describes the job step and is utilized by the initiator in making allocations. It contains the step name, program name, maximum step running time, number of Step Input/Output Tables, step completion code, pointers to the first Step Input/Output Table and the next Step Control Table, and other pointers and status information [IBM 1972a].

Step Input/Output Table: The interpreter creates a Step Input/Output Table for each DD statement in the input stream. This table completely describes the data set, its I/O device requirements, and contains a pointer to the next Step Input/Output Table for the step. The information contained in the table is used by the initiator to build the Task

Input/Output Table and is used by the Allocation and Disposition routines [IBM 1972a].

Supervisor Queue Space: The Supervisor Queue Space is a region of main storage just above the Nucleus. It contains control blocks, lists, and tables required by the control program. The maximum size of this area is specified at system generation time and space is allocated and freed from within this area as required during execution [Flores 1973].

Task Input/Output Table: The Task Input/Output Table is contructed by job management routines. It resides in main storage during step execution and provides information to various I/O routines. The table contains the job name and step name. It also contains an entry for each data set requested for the step. This entry contains such information as label processing insructions, rewind and unloading instructions, unit affinity or channel separation requirements, and volume mounting instructions [IBM 1972b].

Unit Control Block: There is a Unit Control Block for each device on the system. It contains information describing the device characteristics, the current device status, the internal job identification, the protection key of the job, the serial number of the volume assigned to the device, label information, a pointer to the device error routine, and other information required by to the I/O supervisor and job scheduler. Each Unit Control Block contains two segments, a segment containing information common to all devices and a segment containing device peculiar information [IBM 1972b].

APPENDIX B

The macros described below can each be thought of as an interface to a system routine. When the Assembler encounters one of these macros it generates a series of instructions to pass parameters and originate a supervisor call (SVC). At execution time the SVC instruction causes an interrupt. The SVC interrupt handler determines the type of SVC and routes control to the appropriate system routine.

ATTACH: The ATTACH macro causes the control program to create a new task and indicates the entry point in the program. The new task is a subtask of the task which issued the ATTACH macro and it assumes the dispatching priority of the originating task. The issuing program can provide the attached task with a parameter list, an exit routine to be given control when the task terminates, and a control block in which to post its termination. If a usable copy of the program to be given control is not in main storage, it is loaded [IBM 1972c].

DEQ: The DEQ macro causes the control program to remove control of one or more serially reusable resources from the issuing task. The DEQ macro must be used to release every resource obtained through the use of the ENQ macro. If a task attempts normal termination while it still has control of any resources assigned through an ENQ, it will be abnormally terminated [IBM 1972c].

ENQ: The ENQ macro requests that the control program symbolically assign control of one or more serially

reusable resources to the task issuing the macro. Access to a resource is logically, not physically restricted. This means that a task may use a serially reusable resource without using the ENQ macro, but results of doing so may be unreliable. Once control of a resource is symbolically obtained by a task, it remains until that task issues a DEQ macro specifying the same resource [IBM 1972c].

FREEMAIN: The FREEMAIN macro asks the control program to free space that is no longer needed. The space which can be freed is any space obtained through the use of the GETMAIN macro. The space is freed by adding queue elements representing the space to chains recording free areas of main storage. If the area to be freed is a region, the FREEMAIN routine issues a FREEPART macro. Upon completion, control returns to the calling routine [IBM 1972d].

FREEPART: The FREEPART routine is entered from the FREEMAIN routine. Its first function is to use the Task Control Block representing the task to locate the partition queue element of the appropriate region. A FREEMAIN is then issued to release the space in the Supervisor Queue Area occupied by the partition queue element. Next, if the region to be freed is adjacent to an existing free area, it is combined with that area. If not, an element representing the region is added to the chain of free areas. Finally, if there is an initiator waiting for a region it is notified and control returns to FREEMAIN [IBM 1972d].

GETMAIN: The GETMAIN macro is used for all requests for space in main storage. These include requests for regions, space within regions, and space within the Supervisor Queue Area. The GETMAIN routine scans

queues of elements that represent available space to locate the amount of space of the type requested. When the space is found, the affected queues are updated indicating that the space is no longer available and the address of the space is returned to the requester. If the request is for a region the GETMAIN routine issues the GETPART macro [IBM 1972d].

GETPART: The GETPART routine is entered via the GETMAIN routine. It first determines whether there is enough space in the Supervisor Queue Area to initiate the job step and whether there is enough free space in the dynamic area to fulfull the request. If there is not enough space available, the initiator is put into a wait state and control passes to the highest priority ready task in the system. If the requested space is available, the GETPART routine issues a GETMAIN to obtain space in the Supervisor Queue Area for the partition queue element. The GETPART routine then adjusts the pointer in the partition queue element to point to the newly acquired region and returns control to GETMAIN [IBM 1972d].

APPENDIX C

This appendix presents a detailed description of job initiation under OS/360. Readers unfamiliar with OS/360 should refer to Appendix A and Appendix B for explanations of IBM terms and system macro instructions.

The initiator is composed of many modules, each with a specific function. Some of these modules are permanently core resident in the Link Pack Area, others are brought into core only as they are needed. The flow of control depends upon the characteristics of the job being initiated and upon the availability of the resources requested.

The discussion below follows the flow of control through the initiator modules during the initiation of a typical job. Coverage is limited to resource allocation and is not concerned with other housekeeping functions which do not affect deadlock prevention.

A. JOB SELECTION

The Job Selection routine must first determine if there is a job in the input queue which can be initiated by this initiator. To do this it issues an ENQ macro on the queue control record to obtain exclusive control of it. When the queue control record becomes available it is read into core and examined. If there is no job in the input queue a DEQ macro is issued, freeing the queue control record. A message is then sent to the console informing the operator that the initiator is waiting for work. Control passes to the Initiator Wait routine resident in the Link Pack Area which frees the core held by the initiator. When a job is

placed in the input queue the Initiator Wait routine is notified. It then obtains a new region of main storage and passes control to the Job Selection routine.

If there are jobs in the input queue, the highest priority job is selected. The queue control record is updated to indicate the selection and is written back into the queue. Then a DEQ macro is issued permitting access to the queue control record by other routines.

The job control table describing the selected job is read into core. From it is extracted the number of steps in the job and the location of the step control table for each step. The step control table for the first job step to be initiated is now read into core.

The dispatching priority of the initiator is changed to the priority specified for the job. Reducing the initiator's priority forces its contention for system resources (e.g. the CPU) to be at a priority equal to that of the job being initiated.

Next the Job Selection routine reads into core the table containing the name and disposition of all non-temporary data sets required by the entire job. With this information it builds a parameter list for the ENQ macro specifying each data set name and whether its use is exclusive or shareable. When the parameter list is completed it is written into the Supervisor Queue Space and control is passed to the Determine Region Size routine.

B. REGION MANAGEMENT

The modules of the initiator operate in various regions of main storage. When the initiator is first entered it operates in a region obtained for it by the System Task

Control routine. After that, main storage is obtained and released by the initiator modules in the Link Pack Area.

The Determine Region Size routine compares the region size requested by the job with the minimum region size required to initiate a job step. It determines the step region size as the greater of the two. It then stores this region size, along with other information not pertinent to this discussion, into a work table and writes it into the Supervisor Queue Space. The location of the work table is passed to the Get/Free Region routine.

The Get/Free Region routine resides in the Link Pack Area. When it is entered, it frees the old region used by the initiator and issues the ENQ macro for data sets using the parameter list created by the Job Selection routine. If the routine is unable to obtain all of the data sets specified, it issues a DEQ to free those obtained. It then sends a message to the console notifying the operator of the data sets not available. The operator is given the option of retrying the ENQ, cancelling the job, or waiting for the data sets to become available.

When the ENQ is completed, the storage occupied by the parameter list is freed. The GETPART macro is then issued to obtain the region specified in the work table. When it has obtained the region of main storage, control is passed to the Allocation Interface routine which is loaded into the new region.

C. I/O DEVICE ALLOCATION

The Allocation Interface routine releases the core used by the work table, does some routine status checking which need not be discussed, and calls the I/O Device Allocation routine as a subroutine. The I/O Device Allocation routine

is made up of submodules, therefore numerous module names will appear in this section.

Since both the I/O Device Allocation and Termination routines modify the unit control blocks and may be operating concurrently on seperate system tasks, the unit control block table must be protected from multiple accesses. To prevent this the Allocation Entry routine issues an ENQ on the table for the I/O Device Allocation routine. Since the ENQ is issued at the beginning of the routine the effect is to prevent allocation from being performed for more than one job step at a time.

When the Housekeeping routine receives control it issues an ENQ on the unit control blocks in the name of the Termination routine. This prevents termination of any job step while housekeeping is taking place. This routine also issues an ENQ on the unit control blocks to prevent allocation routines for another task from using them during this allocation processing.

The Housekeeping routine analyzes the requests for I/O devices and stores the information necessary for allocation in tabular form for reference by the allocation routines. It also references the system catalogue in order to add volume information to existing tables, such as the Step Control Table and the Step Input/Output Table.

Before the Housekeeping routine passes control to the Allocation Control routine, it issues a DEQ to allow Termination to have access to the unit control blocks while Allocation Control is building tables. The tables built at this time are numerous; a detailed description of each would be impractical and would not contribute to meeting the goals of this paper. Most of the tables are not completed at this time but are filled by other modules. At this point the

space required for each is determined, main storage is allocated, and a control table which points to them is built.

The Demand Allocation routine is entered from the Allocation Control routine. It issues an ENQ on the unit control blocks to prevent access by the Termination routine and then begins to build tables. The first table filled in is the allocate work table. The allocate work table entry contains information describing the data set and other information required for device allocation. This information is obtained primarily from other tables such as the Step Input/Output Table. Using the information now in the allocate work table, a search is made of the unit control blocks for all devices which would satisfy the requirements of the data set. A bit pattern representing these devices is stored in the allocate work table.

Next the channel load table is filled in. For the purpose of allocation, the load on a channel is defined as the number of data sets which are currently accessible through that channel. The channel load table contains an entry for each logical channel in the system.

When the Demand Allocation routine has finished building tables, it then begins making unit assignments. It first makes all assignments for resident direct access volumes. It then determines which devices are not eligible for allocation and eliminates them from the range of consideration. A device is ineligible if it is off-line, contains a resident volume, or has been allocated. The device is eliminated from the range of consideration by setting the bit corresponding to the device to zero in the bit pattern of the allocate work table entry. The final bit pattern represents only devices which can satisfy the request.

Finally, all explicit and implicit requests for specific devices are honored. If there are still outstanding requests at this point, control is passed to the Decision Allocation routine; otherwise, control is passed to the Task Input/Output Table (TIOT) Construction routine.

The Decision Allocation routine first examines each allocation request and determines which units could be used to fulfill them. This decision takes into account requests for channel separation between data sets, requests for data sets to be on the same device, and that data sets passed from prior job steps are already mounted on specific devices. When the range of eligible devices has been reduced as much as possible, allocation begins. Data sets are selected for allocation beginning with the most restrictive requests. When the data set has been selected, devices are considered in the following order: a device on the channel with the greatest number of free units, a device on the channel with the lightest work load, or the first device in the unit control table. The number of free units is determined by consulting the unit control blocks and the channel work load is obtained from the channel load table. When a device is chosen it is eliminated from the list of eligible devices for all other data sets and the next data set is selected for allocation. When allocations have been made for all requested data sets, control is passed to the TIOT Construction routine.

If allocation is impossible because of lack of devices, the operator is given the option of cancelling the job, making a device available which is offline, or waiting for devices to become available. If the wait option is selected, the Wait for Unallocation routine is entered. The Wait for Unallocation routine issues two DEQ's which allow access to the unit control blocks by the Termination routine

and by allocation routines for other tasks. These two DEQ's correspond to the two ENQ's issued in the Housekeeping routine. When a task terminates, freeing its I/O devices, control returns to the Wait for Unallocation routine which issues two ENQ's on the unit control blocks. Allocation is then attempted again.

The TIOT Construction routine builds the task input/output table. The task input/output table is used to provide information to I/O support routines such as OPEN, CLOSE, and EOV. The table contains an entry for each data set requested for the job step. The TIOT Construction routine also allocates direct access storage space on public volumes.

The External Action routine issues mounting instructions to the operator for all volumes required. If an incorrect volume is mounted the operator is again issued a mount message containing the correct volume identification. After all volumes have been verified control is passed to the Allocation Exit routine.

The Allocation Exit routine is entered after all allocations have been successfully made or if an error condition has been detected. If the routine is entered because of an error condition it sets a bit indicating that the job step should be flushed. If the routine is entered after allocation it writes the allocation messages into the output queue. In either case, it issues two DEQ's to allow access to the unit control blocks by allocation and termination routines performing other tasks and returns control to the Allocation Interface routine.

D. ATTACHING TASKS

By the time the initiation process has reached this stage all of the allocations have been completed. What remains is gathering the information needed by the supervisor to run the task. This information is passed via the ATTACH macro.

Upon the completion of device allocation, control returns to the Allocation Interface routine. It tests the condition of the allocation to determine whether or not it was successful; if so, it proceeds. The Table Breakup routine is used to convert the task input/output table into a series of records which are added to the task's input queue entry. Next the job control table and the step control table are written back into the work queue data set.

The Allocation Interface routine now builds the parameter list for the ATTACH macro. It also builds the initiator parameter list which contains pointers to the ATTACH parameter list and other pertinent lists and tables. Control is now passed to the Pre-Attach routine.

The Pre-Attach routine determines the priority at which the job step will be attached and passes control to the Attach routine which issues the ATTACH macro. When ATTACH has been issued, execution of the job step begins.

E. TASK TERMINATION

A task is terminated when it reaches normal completion, abnormally ends execution, exceeds its requested time interval, exceeds its output limit, or is cancelled by the operator. When any of the above conditions is encountered, control is given to the Termination routine which functions as a subroutine of the initiator. The Termination routine

is a collection of modules performing termination housekeeping, step termination, and job termination.

The first module encountered is the Termination Entry routine. It places termination messages into the output queue and if the program ended abnormally it passes control to the Indicative Dump routine. The Indicative Dump routine writes a message to the programmer indicating the user and system completion codes and passes control to the Step Termination Control routine.

The Step Termination Control routine determines whether the step was normally terminated. If not, certain actions are required. In this discussion a normal termination is assummed and control is passed to the Data Set Driver routine.

The Data Set Driver routine reads a step input/output table entry for the step into main storage. It then passes control to the Disposition and Unallocation routine which processes the data set specified by the entry. Control returns to the Data Set Driver which brings in the next step input/output table entry. This loop continues until all entries for the step have been processed; control then returns to the Step Termination Control routine.

The Disposition and Unallocation routine first performs data set disposition as specified in the data definition statement. This includes updating the system catalogue to reflect an addition or deletion of a data set, scratching space on direct access volumes, and/or passing data sets to subsequent job steps. Next an ENQ macro is issued on the unit control blocks and the I/O device corresponding to the data set is unallocated. If the data set can be demounted from the device, a message is sent to the console informing the operator. When unallocation is complete a DEQ is issued

freeing the unit control blocks and control returns to the Step Termination Control routine.

If the job step just terminated is not the last step in the job, control passes to the Step Termination Exit routine which returns control to the calling routine. In this case the calling routine is the initiator which then begins processing the next job step. If the job step just terminated was the last step in the job the Job Termination Control routine is entered.

The Job Termination Control routine examines the job control table to determine if any data sets have been passed or retained which were not referenced by a subsequent step. If there are, the Disposition and Unallocation routine is used to process them. When all data sets have been processed control passes to the Job Termination Exit routine.

The Job Termination Exit routine places the last termination message into the output queue, completes the output queue entries, and issues a job ended message to the operator. The initiator is now free to process another job and the entire initiation process begins again.

APPENDIX D

The entries in this appendix describe MTS elements used in communications between the supervisor and the Paging Drum Processor. The source of information for all five entries is [Alexander 1971].

Page Control Block: There is a page control block for every virtual memory page in the system. The page control block is always on a linked list of pages owned by the task and may also be on one of the four system queues listed below. It contains the status of the page. The fields of the page control block are: the address of the next PCB for the same task, the main storage address of the page, the virtual memory address of the page, the status of the PCB, the address of the PCB on the system queue, the number of pages in the same block, the address of the Task Control Table for the task owning the page, the number of lock requests for the page, a scratch area to be used by the supervisor, the storage key, the status of the page, and the auxiliary storage address of the page.

Page In Queue: This is a queue of all PCB's for pages which have been requested by the supervisor, but which the Paging Drum Processor has not yet started to read in.

Page In Complete Queue: This is a queue of all PCB's for pages which have been read in by the Paging Drum Processor, but of which the supervisor is not yet aware.

Page Out Queue: This is a queue of all PCB's for pages in main storage which can be paged out if more space is required. The PCB's are arranged in order of desirability of removal.

Release Page Queue: This is a queue of PCB's for all pages which have been released by their tasks, but which have not yet been released by the Paging Drum Processor.

APPENDIX E

entries in this appendix describe supervisor calls y the Paging Drum Processor. The source of tion for all five entries is [Alexander 1971].

ues: This supervisor call is used by the Paging Drum ocessor to get the Page In Queue and the Release Page eue.

l Page: This supervisor call is used by the Paging um Processor to request a main storage block. If a ock is available it is allocated to the page and the dress is returned in the page control block. If no ock can be allocated a condition code so indicates.

te Page: This supervisor call is used by the Paging um Processor to request the number of pages required fill the empty slots on the paging drum for the evolution being scheduled. The supervisor will return to the number of pages requested, depending on their vailability and how full memory is.

eal Page: This supervisor call is used by the Paging rum Processor to inform the supervisor that a block of ain storage is now available for reallocation. It is lso used by the supervisor to notify the PDP that the age was reclaimed by its task while it was being ritten out to the drum. In this case the PDP marks he copy of the page on auxiliary storage as invalid.

it: This supervisor call is used by the Paging Drum rocessor to inform the supervisor that it has no more

work tc do. Therefore the next time the PDP is required it will have to be restarted by the supervisor.

BIBLIOGRAPHY

[Alexander 1971] Alexander, M. T., "Time Sharing Supervisor Programs", Advanced Topics in Systems Programming, University of Michigan Engineering Summer Conference, 1971, p. 1-64

[Alexander 1972] Alexander, M. T., "Organization and Features of the Michigan Terminal System", 1972 Spring Joint Computer Conference, (May 1972), p. 585-591

[Boettner 1970] Boettner, D. W. and Alexander, M. T., "MTS - Michigan Terminal System", SIGOPS Operating Systems Review, 4,4 (Dec 1970), p. 6-19

[Clark 1966] Clark, W. A., "The Functional Structure of OS/360: Part III Data Management", IBM Systems Journal, 5,1 (1966), p. 30-51

[Coffman 1971] Coffman, E. G.; Elphick, M. J.; and Shoshani, A., "System Deadlocks", Computing Surveys 3,2 (June 1971), p. 67-78

[Dijkstra 1965] Dijkstra, E. W., "Solution of a Problem in Concurrent Programing Control", Comm. ACM 8,9 (Sept 1965), p. 569

[Flores 1973] Flores, I., Operating System for Multiprogramming with a Variable number of Tasks, Allyn and Bacon, 1973

[Habermann 1969] Habermann, A. N., "Prevention of System Deadlocks", Comm. ACM 12,7 (July 1969), p. 373-377,385

[Havender 1968] Havender, J. W., "Avoiding Deadlock in Multitasking Systems", IBM Systems Journal 7,2 (1968), p. 74-84

[Holt 1971] Holt, R. C., On Deadlock in Computer Systems, Ph. D. Thesis, Cornell University, 1971

[Holt 1972] Holt, R. C., "Some Deadlock Properties of Computer Systems", Computing Surveys 4,3 (Sept 1972), p. 179-196

[IBM 1971] IBM System/360 Operating System Introduction, IBM form number GC28-6534-3, June 1971

[IBM 1972a] IBM System/360 Operating System MVT Job Management Program Logic Manual, IBM form number GY28-6661-9, March 1972

[IBM 1972b] IBM System/360 Operating System: System Control Blocks, IBM form number G28-6628-8, March 1972

[IBM 1972c] IBM System/360 Operating System Supervisor Services and Macro Instructions, IBM form number GC28-6646-6, March 1972

[IBM 1972d] IBM System/360 Operating System MVT Supervisor IBM form number GY28-6659-6, March 1972

[U of Mich 1972a] "Details on Using Shared Files in MTS", University of Michigan Computing Center Memo M228, (Dec 1972), p. 1-5

[U of Mich 1972b] "Shared Files - It's the Real Thing", University of Michigan Computing Center Newsletter, 2,15 (Oct 1972), p. 1-5

[Witt 1966] Witt, B. I., "The Functional Structure of OS/360: Part II Job and Task Management", IBM Systems Journal, 5,1 (1966), p. 12-29

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Professor G. L. Barksdale, Jr, Code 72 Computer Science Group Naval Postgraduate School Monterey, California 93940	1
4.	Capt Leslie R. Conklin, USMC 1058 Halsey Drive Monterey, California 93940	1
5.	Chairman, Computer Science Group, Code 72 Naval Postgraduate School Monterey, California 93940	1
6.	LTJG R. H. Brubaker, Jr, Code 53BH Computer Science Group Naval Postgraduate School Monterey, California 93940	1

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Naval Postgraduate School Monterey, California 93940	Unclassified
	2b. GROUP

REPORT TITLE

A Study into the Problem of Deadlock

DESCRIPTIVE NOTES (Type of report and inclusive dates)

AUTHOR(S) (First name, middle initial, last name)

Leslie R. Conklin

REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
June 1973	53	19

CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO.	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	

0. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

1. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	Naval Postgraduate School Monterey, California 93940

3. ABSTRACT

This paper presents a study of the problem of system deadlock with emphasis on the allocation of serially reusable resources. The initial area of concentration is on a definition of system deadlock and a presentation of proven methods of handling deadlock. After the discussion of current theory, two major operating systems are examined. An overall description of the functioning of OS/360 is presented followed by a detailed discussion of the procedures used by OS/360 to allocate serially reusable resources. The Michigan Terminal System is then discussed, outlining the techniques of resource allocation employed. The paper is concluded by correlating the methods used in OS/360 with those of MTS.

DD FORM 1 NOV 68 1473 (PAGE 1)

S/N 0101-807-6811

Security Classification

A-31408

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
ichigan Terminal System						
erating System/360						
ource allocation						
ially reusable resource						
tem deadlock						

Thesis
C7023 Conklin
c.1 A study into the
problem of deadlock.